Study on Memory Hierarchy Optimizations

Sreya Sreedharan,Shimmi Asokan

Abstract—Cache is an important factor that affects total system performance of computer architecture. Due to the ever increasing performance gap between the processor and the main memory, it becomes crucial to bridge the gap by designing an efficient memory hierarchy capable of reducing the average memory access time. Many of the recent studies in improving performance of cache have focused on minimizing the cache miss rates. Cache miss rate can be reduced by optimizing data or instruction cache. Data is too large to be kept in cache so optimization of data cache is necessary since it has to be moved between memory and cache frequently. A way to decrease data cache miss is to restructure the code. This paper explains various restructuring techniques and analyzing of cache miss rates using valgrind tool.

Index Terms—Cache, Cache Miss, Cache miss rate, Average Memory Access Time

1 INTRODUCTION

The main memory used in personal computers is dynamic RAM. It is slower and holds data as long as power is applied. So static RAM is introduced which is faster when compared to dynamic RAM. A memory cache is a static RAM introduced between processor and main memory in order to store copies of data from frequently used main memory locations. It is smaller and expensive when compared to main memory. When a program needs to access data from the disk, it first checks the cache. Data is transferred between memory and cache in blocks of fixed size known as cache lines or cache blocks. If the required block is found within a cache, a cache hit occurs. If the required block is not found, a cache miss occurs. If miss occurs, the block must be obtained from the main memory.

Compulsory, capacity and conflict misses are the three types of cache misses. In order to make space for the new entry, cache may have to evict one of the existing entries. The method that it chooses the entry to evict is known as replacement policy. There are various replacement policies. In least-recently used approach, least recently used blocks are discarded. In most-recently used approach, the most recently used blocks are discarded. This approach can be used in situations where older items are most likely to be accessed. The best approach would be to discard the data that will not be needed for the longest time in future, which uses Belady's algorithm. But this is not practical since it is impossible to predict the future references. Another approach is known as least-frequently used in which those blocks which are used less frequently are discarded first.

- *SreyaSreedharan is currently pursuing Mtech in department of CS inRajagiri school of* engineering and technology, kerala ,India, E-mail: twindudes@gmail.com
- *Co-AuthorShimmiAsokan is working as Assrstant professor in depertment of CS in Rajagiri school of engineering and technology, kerala ,India,E-mail:shimmi_a@rajagiritech.ac.in*

The very first access to the requested block results in a miss. Such a miss is known as compulsory miss or cold start miss. Under this type the block that is needed must be bought into the cache from the main memory.
A capacity miss occurs when a block that is requested was there in the cache earlier, but was discarded due to not enough capacity of the cache to hold all the blocks needed for the current execution. It occurs due to the finite size of the cache, regardless of block size or associativity. In direct mapped or set-associative caches even if the cache has enough space to hold blocks, blocks are discarded in order to make space for another block in the set. If request comes for such a discarded block, conflict miss or interference miss is said to occur.

Cache works on the principle of locality of reference. There are two types of locality of reference. They are spacial and temporal. If a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future. This approach is known as spacial locality of reference. If at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future. This approach is known as temporal locality of reference.

In a system with virtual memory, the virtual address generated by the processor is translated to physical address and is used to access the cache. A page table is used to store the virtual address to physical address mappings. To speed up virtual address translation, the system stores recently used translations in the translation look-aside buffer (TLB), which is a separate cache. The search key is the virtual address and corresponding physical address is obtained from the TLB.

A memory hierarchy which includes cache improves performance. Cache performance can be measured in terms of Average Memory Access Time, which is given by

Average Memory Access Time= Hit Time + Miss Rate * Miss Penalty

where, Hit time is the time taken to hit in the cache, Miss Rate is the number of accesses that miss/Total no. of accesses and Miss Penalty is the time needed to service a miss [8] .

Efficiency of caches depend on the spacial and temporal properties of programs. In order for the program code to efficiently utilize the cache ,program can be restructured. Restructuring may aim at accessing contiguous locations in memory by changing data access order or by providing maximum reuse opportunities for cache. Changing the data access order ensures the access of data in the order in which they are stored. Reuse of cache can be done by utilizing the data already being fetched from memory to the maximum before being evicted from the cache. This kind of restructuring reduces the number of times each line has to be fetched into the cache. Valgrind tool helps in analyzing cache misses in a program and the effect of code restructuring in improving cache misses can be studied.

2 Related Work

There have been several approaches to analyze various cache improvement techniques. [8] Provides an excellent discussion on this topic. The various techniques for improving cache performance focusing on reducing the miss penalty, reducing the miss rate and reducing the time to hit in the cache were discussed in detail. Techniques under reducing miss penalty involve introducing multi-level caches, considering read miss before write miss, critical word first and merging write buffers. techniques under reducing miss rate involve large cache size, large block size, increasing associativity, victim cache, pseudo associativity and compiler optimization techniques. Time to hit in the cache can be reduced by using small cache or by avoiding address translation.

[15] Discusses two of the cache performance improvement techniques such as sub-block placement and victim cache concept. [14] Discusses the data and instruction cache optimization and various loop transformations in detail. [5] Discusses loop tiling in detail and [1] discusses a minor variation of loop tiling.[6] Provides the necessary details of memory hierarchy involving cache and the way cache access is done in detail. [13] Explains loop interchange in detail with an example. [7] and [10] details various loop transformations. In this paper we attempt to analyze restructuring of program code and its effect on cache performance.

3 Sections

Caches are organized as a collection of cache blocks or lines. A cache block is the unit of data transfer to/from an underlying layer in the memory hierarchy .The mapping between memory blocks and cache blocks is an important design issue. There are three general approaches for the mapping of a block to the cache. They are direct mapped cache, fully associative cache and set associative cache.

Direct mapped cache: This is the simplest approach. In this approach, the main memory block index 'k', get mapped on to 'k mode n' of cache memory, where 'n' is the number of blocks in the cache. Direct mapping might lead to greater page faults which can be reduced by associative mapping.

Fully associative cache: This is the fastest mapping technique. In a fully associative cache a memory block can be placed in to any of the cache blocks. This technique consumes more number of tag bits.

Set associative cache: In a k-way set-associative cache, cache blocks are divided into sets. Each set contain 'k' blocks. A memory block can be placed in any block of a particular set. Mapping is done using the function 'k mode s', where 'k' is the main memory block index and 's' denote the number of sets. This technique requires less number of tag bits compared to fully associative mapping and more number of tag bits compared to direct mapping.

As an example, let us assume that we have a direct-mapped cache and two scalar variables 'a' and 'b' that are being accessed by a program in the order "a, b, a, b, a, b, a, b". If these two variables are laid out in memory such that they belong to two different memory blocks that map to two different cache blocks. Then this access pattern leads to only two misses. On the other hand, suppose the memory layout is such that 'a' and 'b' map to two memory blocks that in turn get mapped to the same cache block. Then the same access pattern may lead to all eight accesses being a miss. Therefore, layout of data in memory can play a crucial role [12].

4 Optimization Technique

Most systems follow harvard architecture which is shown in figure 1. Harvard architecture uses 2 level cache memory in which level 1 cache is a split cache consisting of data cache and instruction cache. Level 2 cache is a unified cache. Harvard architecture allows the processor to fetch instructions from the instruction cache and data from the data cache simultaneously .Instruction cache optimization is not given much importance because most of their execution time is spent in small computational kernels based on loop nests[10]. But data is too large to be kept in a higher level of memory hierarchy such as cache and thus optimization of data cache has to be focused on.

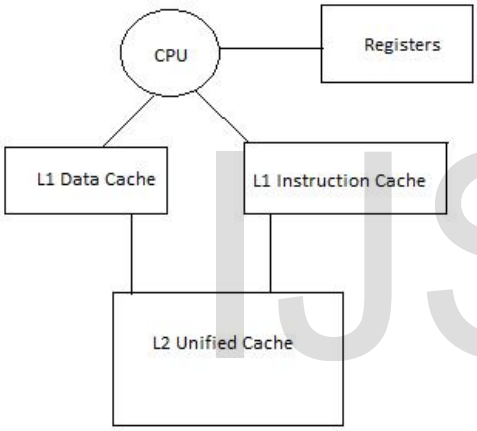

Fig 1: A typical memory hierarchy

Data cache optimization can be further classified on to data access optimization and data layout optimization. Data access optimization is restructuring the code by changing the order of execution of the program. Loop transformations fall into the category of data access optimization. Loop transformations include loop interchange, loop fusion and loop tiling.

i) Loop Interchange

Programs have nested loops that access data in memory in non sequential order. Simply exchanging the nesting of the loops can make code access the data in the order in which they are stored. It reduces misses by improving spatial locality. It specifically helps in decreasing compulsory miss.

ii) Loop Fusion

This technique combines two independent loops that uses the same variables. Some programs have separate sections of code that access the same array or those that perform different computation on common data. Fusing multiple loops into a single loop allows the data in cache to be used repeatedly before being swapped out. Loop fusion reduces misses through improved temporal locality.

iii) Loop Tiling

Loop tiling tries to reduce misses via improving temporal locality. The goal is to maximize accesses to the data being loaded in to the cache before the data are replaced. Tiling decreases capacity miss. It operates on sub matrices or blocks unlike loop interchange. Loop tiling can be also termed as blocking.

5 Results and Discussion

Programs 1.1 and 1.2 given below demonstrates loop interchange. The first program is the loop interchanged code and its data cache miss rate is less when compared with the second program. This is because program 1.1 access data in the order in which they are stored assuming c follows row- major access.

Program 1.1: intmain(void)

```
{
int h, i, j, a[1024][1024];
for (h = 0; h < 10; h++)
for (i = 0; i< 1024; i++)
for (j = 0; j < 1024; j++)
a[i][j] = 0 ;
return 0;
}
```

Program 1.2: intmain(void)

```
{
int h, i, j, a[1024][1024] ;
for (h = 0; h < 10; h++)
for (i = 0; i< 1024; i++) for (j = 0; j < 1024; j++)
a[j][i]=0;
return 0;
}
```

In order to profile loop interchange effects we use valgrind tool. Valgrind has an associated tool ,Cache Grind which is a cache simulator. Loop interchange operation reduces first level data cache miss rate (D1d miss rate) from 19.9% to 1.2%[13].

Program 2.1 and 2.2 given below demonstrates loop fusion. Program 2.1 is the program before applying loop fusion and program 2.2 is on applying loop fusion[11]. This operation reduces D1d misses from 1.7% to 0.9%.

Programs 3.1 is the program before applying loop tiling and 3.2 demonstrates loop tiled program[11]. This operation reduces D1d miss rate from 9.6% to 0.0%.

Program 3.1: void main()
{
inti,j;
int a[1024][100] int b[1024][100];
for (i = 0; i< 1024; i++) for (j = 0; j < 1024; j++) a[i][j] = a[i][j] + b[j][i];
}

Program 3.2: void main()
{
inti,j;
int x, y, c[1024][100]; int b[1024][100];
for (i = 0; i< 1024; i += 2) for (j = 0; j < 1024; j += 2) for (x = i; x <i + 2; x++) for (y = j; y <j + 2; y++) c[x][y] = c[x][y]+ b[x][y];
}

Figure 2 and 3 shows the benefit of using loop transformations.In fig. 2 level 1 data cache misses are analyzed and in fig. 2 last level data cache misses are analyzed.

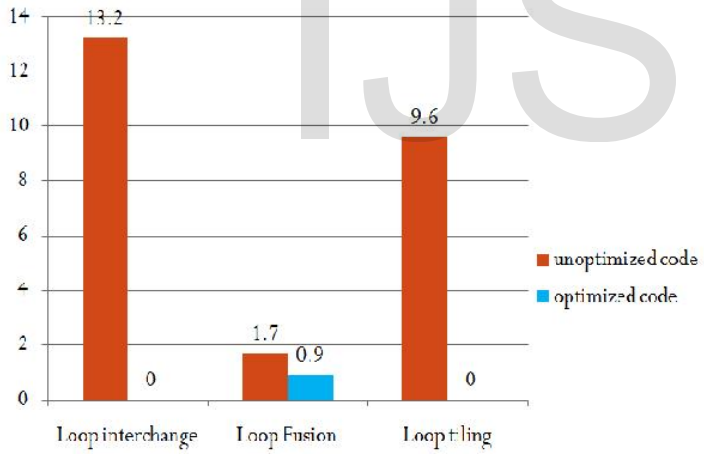

Fig 2.Improvement in D1 Miss Rate using Loop transformations

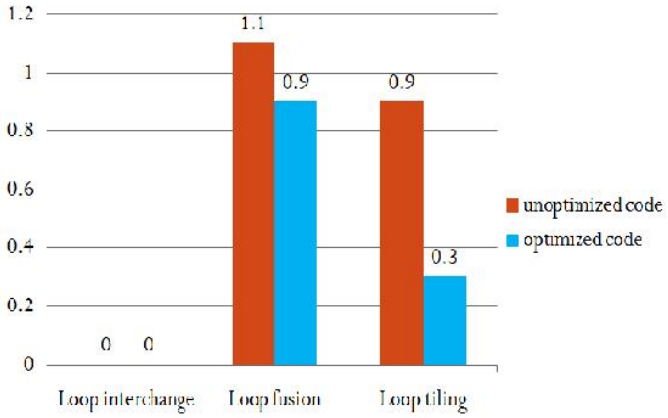

Fig. 3: Improvement in LLd Miss Rate using Loop transformations

6 Conclusion

Programs were run in valgrind tool and cache misses were analyzed. Loop interchange significantly reduces level 1 data cache (D1) miss rate. Loop fusion reduces the last level data miss rate (LLd) and D1 miss rate slightly. Loop tiling significantly reduces last level data miss rate and D1 miss rate which greatly improves performance.

References

[1] PreetiRanjan Panda, Nakamura, Nikil.D.Dutt, AlexandruNicolau, "Augmenting Loop Tiling with Data Alignment for improved cache performance", IEEE Transactions on computers, Vol. 48, NO. 2, February 1999.

[2] AnantAgarwal and Steven Pudar, "Column-Associative Caches: A Technique for reducing miss rate in direct-mapped caches", In Proc. 20th annual International Symposium on Computer Architecture, Pages 179-190, May 1993.

[3] Scott McFarling, "Cache Replacement with Dynamic Exclusion", Western Research Laboratory 250 University Avenue Palo Alto, California November 1991.

[4] Steven P. Vanderwiel, David J. Lilja, "Data Prefetch Mechanisms", ACM Computing Surveys, VOl.32,Issue 2, June 2000.

[5] Monica S. Lam, Edward E. Rothberg and Michael E. Wolf, "The Cache Performance and Optimization ofBlocked Algorithms", Proceedings of fourth International Conference on Architectural support for programming languages and operating systems, CA April 1991.

[6] William.L.Lynch, "The interaction of Virtual Memory and Cache Memory", Technical Report CSL-TR-93-587,October 1993.

[7] Kathryn.S.McKinley, Steve Carr, Chau-Wen Tseng "Improving data locality with loop transformations,
IEEE Vol. 18, No. 4, July 1996.

[8] John.L.Hennessy, David.A.Patterson, "Computer Architecture: A Quantitative Approach, Fourth Edition, Morgan Kaufmann Publishers".

[9] Blas Cuesta, Alberto Ros, Marı´aE.Go´mez, Antonio Robles, and Jose´ Duato, "Increasing the effectiveness of directory caches by avoiding the tracking of Noncoherent memory blocks", IEEE Transactions on computers, Vol. 62, No.3,March 2013.

[10] Markus Kowarschik1 and Christian We, "An Overview of Cache Optimization Techniques and Cache Aware Numerical Algorithms", Springer ,Algorithms for Memory Hierarchies, LNCS 2625, pp. 213-232, 2003.

[11] http://www.vi-hps.org/upload/projects/hopsa/hopsa-nov12-threadspotter.pdf

[12] Wei Ding, Jun Liu, MahmutKandemir and Mary Jane Irwin, "Reshaping cache misses to improve row buffer locality", IEEE 2013.

[13] NikolayPavlovich Laptev, "Analysis of cache architectures", Department of Computer Science – University of California Santa Barbara.

[14] Steven. S. Muchnick, "Advanced compiler Design Implementation", Morgan Kaufmann Publishers.

[15] UmangChoudhary, Pratik Phadke, Vasundhara-Puttagunta, SupreethUdayashankar, "Analysis of Sub-block Placement and Victim Caching Techniques".